

# Travail de Bachelor

Système expert pour adaptation de posologie médicamenteuse.  
Expert System for drug dosage adjustments

Étudiant :	Max Caduff
Travail proposé par :	Yann Thoma REDS route de Cheseaux 1 1401 Yverdon-les-Bains
Enseignant responsable :	Yann Thoma
Expert:	François Veuve
Année académique :	2019-2020

Yverdon-les-Bains, le 09.09.2020

Département: TIC  
Filière: Informatique et systèmes de communication  
Orientation: Ingénierie Logicielle  
Étudiant: Max Caduff  
Enseignant responsable: Yann Thoma

## Travail de Bachelor 2019-2020

Système expert pour adaptation de posologie médicamenteuse.  
Expert System for drug dosage adjustments

---

### Résumé publiable:

In the medical field, when a drug is given to a patient, the concentration it reaches in the blood is different from a patient to another, and can be influenced by various parameters. Pharmacokinetics is a branch of pharmacology that studies the evolution of such drugs concentration in the body. It uses different models to describe the absorption and elimination of foreign substances, and can hence estimate the concentration of the taken dose in the blood at a specific time.

Some software has been created to help pharmacologists do these computations, Tucuxi is one of them, developed at HEIG-VD, which has 2 version: one that needs the doctor to provide a lot of information by hand and specify what he wants to calculate but produces a nice output, and the second that has a more direct input, but some requests still needs to be created, and the results are unreadable, being an XML file.

TucuXpert is a new program using Tucuxi's computing core, that is able to determine alone what computations can be performed depending on which informations are available on the patient (those could be provided by the hospital database for example), and present the results nicely in an HTML file, that can be opened with any browser. It is notably able to calculate blood concentration predictions with the associated percentiles, suggest dosage adjustments or perform some checks on the provided samples if any. It also uses a library to create graphs from the raw points returned by the core and includes them in the report.

Étudiant :

Max Caduff

Date et lieu :

.....

Signature :

.....

Enseignant responsable :

Yann Thoma

Date et lieu :

.....

Signature :

.....

# Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 09 septembre 2020

# Authentication

Je soussigné, Max Caduff, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Renens, le 09 septembre 2020.

Max Caduff

# Table of Contents

I. Introduction.....	5
II. Existing software and needs .....	6
III. Specification .....	8
Input:	8
Request creation:	11
Output:	13
IV. Implementation.....	15
tucuxpert.cpp:	15
TucuxpertHandler:	15
XpertData:	16
QueryCreator:	16
RequestCreator:	17
DrugModelSelector:	18
HtmlCreator:	19
GraphCreator:	20
ILanguage:	21
EnLanguage:	21
Tools:	21
V. Examples of outputs.....	22
Simple case with no treatment:	22
Case with a treatment and no samples, showing multiple adjustments:	23
Case with samples including one outside the range:	24
VI. Difficulties encountered .....	25
VII. Possible improvements .....	26

# I. Introduction

In the medical field, when a drug is given to a patient, the concentration it reaches in the blood must be comprised between some predefined thresholds to be simultaneously efficient and not dangerous. This is tricky because the concentration profile in the blood is not linear and depends on multiple factors, each body dealing slightly differently with such substances, absorbing and excreting them at a different pace. To know if the targets are always respected, the best would be to take multiple blood samples at regular intervals, analyze them in a lab, and make a regression to get a precise curve, but it requires too many handling, currently the blood concentration is measured at most once a day, usually only once, and it then requires some computations to estimate the concentration profile.

Pharmacokinetics is a branch of pharmacology which studies the evolution of such drugs concentration in the body. It uses different models to describe the absorption and elimination of foreign substances, and can hence model the evolution of the taken dose in the blood, knowing the moment it was taken. Since everybody reacts differently, if we know the exact concentration at a certain time, some formulas can be used to refine the prediction.

Those computations being relatively complex and time-taking, they have already been automated by softwares. The aim of this project is to create a program that exploits the computing core of an existing concentration prediction software called Tucuxi, developed by the REDS institute at HEIG-VD. This new program should be able determine which useful predictions can be computed based on the provided informations, and present the results as a human-readable HTML report.

This document will present the actual state of this software, what are the reasons that led to this work and what should be implemented to carry it out. Then, a review of all created classes will be presented along with design choices, in the order of the program flow. Finally, some output examples will be shown, and the difficulties encountered for this project will be evoked as well as the possible improvements.

## II. Existing software and needs

In the existing software, the main part is the computing core, written in c++, which is used to do all supported computations. Those computations includes calculating a first dose for a new medication, adjusting an existing dosage, predicting the concentration at a specific time and calculating the points for a prediction curve or percentiles. They can be made with more or less infos on the patient, if some are missing, default values will be used.

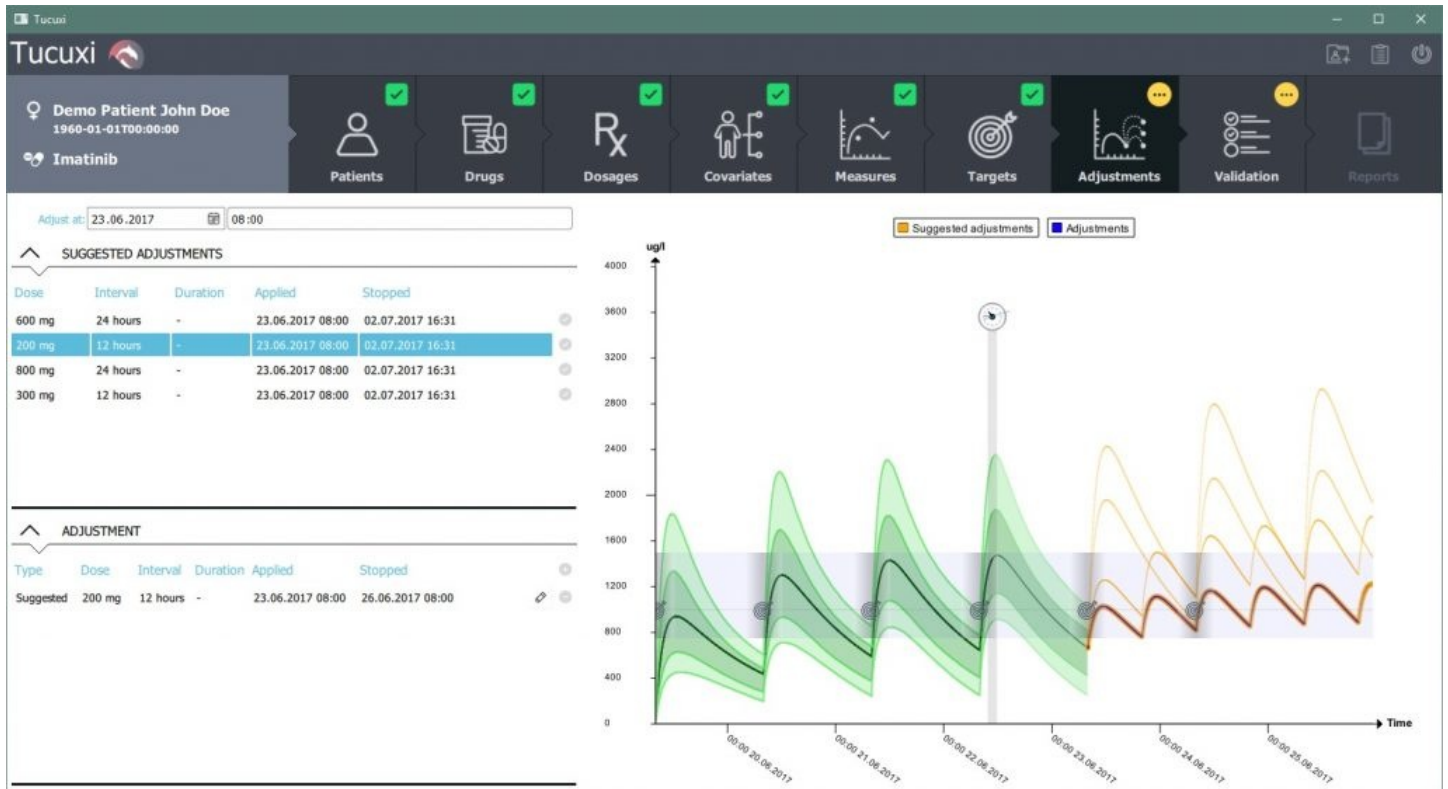
To perform the right computations, the core also needs informations from an XML drug file (also referred to as drug model) for each request, each drug having specificities. For example, the concentration curve of most drugs is wanted to oscillate around an ideal value, but for some others the efficiency is characterized by the presence of a peak of concentration at regular intervals, which should rapidly drop to avoid bad side effects. Some drugs have multiple drug files, each one being created from the results of a clinical study, and is suitable for a population with various characteristics in a certain range.

Some of the specific computational parts are included in those drug files, as a javascript subset, to be executed at runtime.

The other software parts using the core are:

- a Command Line Interface (CLI), taking as input the drug file and an XML file with the patient's data and the request(s), and outputting the computations results as another XML file.
- a Graphical User Interface (GUI), requiring the user to fill all fields manually, and outputting a readable report with the computation result the user asked for.
- a REST server, having the same functionalities as the CLI.

These programs are working well but they all require the user to specify exactly what he wants them to calculate (first dose, dosage adjustment, prediction from date x to y, etc) and to specify the drug model. The GUI version have multiple tabs with numerous fields to fill manually, but produces a very nice output that can be saved as pdf, and the CLI's input is more direct but the requests still needs to be created and its output is not very reading-friendly (XML), especially to read the tables of predicted values; so some needs have been expressed to have a version that is just fed with the patient's data at once without asking any request, so the informations could be entirely provided by a database, then the program should automatically select a drug model, determine what computations can be made with the provided data, launch them, perform some checks on the results, and finally generate an HTML report with graphs that can easily be opened in any browser. This work will try to answer those needs. The main advantage of this new program compared to the existing ones is its ability to generate the requests and select the drug model alone, hence its given name « Tucuxpert ». The type of created requests will be developed in the next chapter.



A GUI screenshot



## III. Specification

### Input:

The program's input will be an XML file, as in the CLI and the REST server, this format offering the advantage that its structure is well defined, and it can be generated automatically by some database.

The input format could also have been JSON, but since the existing input file's structure fits well our needs and the specification is already done, it will be mostly reused, so by using the same input format the parsing class can be partly reused too. The existing format defined for the input files is .tqf (Tucuxi Query File), it will be reused for Tucuxpert input files.

The actual existing structure (without details) is the following:

```
<query>
  <queryId></queryId>
  <clientId></clientId>
  <date></date>
  <language></language>
  <drugTreatment>
    <patient>
      <covariates></covariates>
    </patient>
    <drugs>
      <drug>
        <drugId></drugId>
        <activePrinciple></activePrinciple>
        <brandName></brandName>
        <atc></atc>
        <treatment></treatment>
        <samples></samples>
        <targets></targets>
      </drug>
    </drugs>
  </drugTreatment>
  <requests></requests>
</query>
```

The most important parts are inside the drugTreatment tag: it contains all informations about the patient (age, weight, size, etc) in the covariates tag, the treatment tag contains informations about the actual prescribed treatment if any, samples contains optionals blood samples measures from the patient, and the targets tag can be used to specify different blood concentration targets desired for this specific patient instead of the default ones. The computations launched by Tucuxpert will be based on the content of those fields, instead of relying on the requests tag, which used to contain the precise computations to be performed. This tag and its children will then be removed, the strategy to determine

what to compute is developed below. The other difference with the TucuCLI's XML specification is that a part containing optional administrative data that are handy for the report generation will be added, following the same format than in Nadir Benallal's Bachelor thesis, who worked on the same project.

The complete new XML structure for the input files will then look as follows:

```
<query>
  <queryId>123_ch.heig-vd.Tucuxi.vancomycin_neonate</queryId>
  <clientId>314</clientId>
  <date>28.01.2020</date>
  <language>en</language>
  <drugTreatment>
    <patient>
      <covariates>
        <covariate>
          <covariateId>height</covariateId>
          <date>25.12.2019</date>
          <value>175</value>
          <unit>cm</unit>
          <dataType>double</dataType>
          <nature>continuous</nature>
        </covariate>
      </covariates>
    </patient>
  <drugs>
    <drug>
      <drugId>vancomycin</drugId>
      <activePrinciple>vancomycin</activePrinciple>
      <brandName>somebrand</brandName>
      <atc>J01XA01</atc>
      <treatment>
        <dosageHistory>
          <dosageTimeRange>
            <start>2018-01-09T14:29:00</start>
            <end>2018-01-10T01:53:00</end>
            <dosage>
              <dosageLoop></dosageLoop>
              or
              <DOSAGE_CHOICE></DOSAGE_CHOICE>
            </dosage>
          </dosageTimeRange>
        </dosageHistory>
      </treatment>
    <samples>
      <sample>
        <sampleId>1</sampleId>
        <sampleDate>2018-01-10T14:41:54</sampleDate>
```

```
<concentrations>
  <concentration>
    <analyteId>vancomycin</analyteId>
    <value>23</value>
    <unit>mg/l</unit>
  </concentration>
</concentrations>
</sample>
</samples>
<targets>
  <target>
    <activeMoietyId>vancomycin</activeMoietyId>
    <targetType>residual</targetType>
    <unit>mg</unit>
    <min>20</min>
    <best>25</best>
    <max>30</max>
    <inefficacyAlarm>15</inefficacyAlarm>
    <toxicityAlarm>50</toxicityAlarm>
    <mic></mic>
  </target>
</targets>
</drug>
</drugs>
</drugTreatment>
<administrativeData>
  <mandator>
    <person>
      <personalContact>
        <id></id>
        <title></title>
        <firstName></firstName>
        <lastName></lastName>
        <address></address>
        <phone></phone>
        <email></email>
      </personalContact>
      <instituteContact>
        <id></id>
        <name></name>
        <address></address>
        <phone></phone>
        <email></email>
      </instituteContact>
    </person>
  </mandator>
  <patient>
```

```
<person>
  <personalContact>
    <!-- see above-->
  </personalContact>
  <instituteContact>
    <!-- see above-->
  </instituteContact>
</person>
</patient>
<clinicalData>
  <data>
    <key> key1 </key>
    <value> val1 </value>
  </data>
</clinicalData>
</administrativeData>
</query>
```

The drugId tag and its parents are mandatory, since nothing can be computed without knowing what drug is prescribed, but the patient, treatment, samples and targets tags are all optionals, and different computations will be made depending on which are given. However, having a sample without a treatment makes no sense, the program will terminate with an error in this case. The covariates defined in the drug file have a default value, those will be used if the corresponding covariate is not found in the input file. The default and provided values will be summarized in the report, and missing values will be highlighted to emphasize the fact that default values must have been used.

## Request creation:

Each request for the core needs a drug model, since some drugs have multiple drug files (corresponding to different studies on various population), one need to be selected, and the same will be used for all requests. Drug models have embedded constraints to filter on what kind of patient the model can be applied, they will therefore be first filtered by those constraints, but since there is no way to know which is the best among the valid ones, multiple options have been explored (more details below) to finally select the model with the highest number of fully compatible constraints.

The core can perform 3 type of computation, depending on the data provided:

- population, standing for default population values, when no data on the patient is available.
- apriori, which can be used when some patient covariates are provided, to refine the computation.
- aposteriori, used when a drug treatment and samples are provided, it also refines the computation.

This information must be supplied with each request, and will be the same for all of them.

The computing core can compute 5 type of requests, but only 4 will be used in Tucuxpert:

- ComputingTraitAdjustment: calculates one or multiple dosage prediction from the given parameters
- ComputingTraitConcentration: calculates the values of the estimated blood concentration from the treatment.
- ComputingTraitPercentiles: calculates the points for the asked percentiles.
- ComputingTraitAtMeasures: calculates the estimated value at the exact time of the sample.

The unused type is ComputingTraitSinglePoints, which only calculates values at the supplied times.

Now let's see what kind of queries can be made with those objects depending on the fields provided in the XML input file:

### First Dose:

If the field treatment is absent or empty, it means that the patient has no current dosage, so a first dose will be calculated for every available dosage, and the corresponding prediction curves will be shown to help the doctor choose the best option. The object used here is ComputingTraitAdjustment, which can calculate a dosage even with no data on the patient. The prediction type of this request can be *population* or *apriori*.

### Prediction curve and percentiles:

If a treatment is given, then the corresponding estimated blood concentration curve can be calculated, along with percentiles so they can be compared on a graph. These results are computed using ComputingTraitConcentration and ComputingTraitPercentiles. The type of computation for these requests can be *population*, *apriori* or *aposteriori*.

### Dosage adjustment:

A dosage adjustment can be asked as long as the treatment field present, to check if a better dosage is available. Again all type of computation can be used (*population*, *a priori*, *a posteriori*). The dosage adjustments will be suggested in every available dosage fitting the targets along with the corresponding curves, to let the doctor choose the one he prefers. The object used is the same as for a first dose, a ComputingTraitAdjustment.




### Samples values check:

If one or more samples are available in the treatment, a ComputingTraitAtMeasures object will be used to retrieve the predicted concentration at the precise sample time. This is useful since the data points in ComputingTraitConcentration are calculated every 5 minutes, it allows to compare directly the values to percentiles instead of checking the lower and higher bounds. This computation type is in all cases *aposteriori*.

The following table summarizes the different computations performed depending on which tags are provided:

computation performed \	field presence	treatment	samples
first dose			
prediction curve and percentiles			
dosage adjustment			
sample value likelihood			

Caption:

	field is not present		field is present		field can be present or absent
---	----------------------	---	------------------	--	--------------------------------

this table summarizes the type of computation performed depending on the provided fields. The caption is the same as the previous table:

type of computation \	field presence	treatment	samples	covariates	targets
computation done for the typical patient					
computation done a priori					
computation done a posteriori					
computation uses personalized targets					
computation uses default targets					

## Output:

Once the results have been obtained, some sanity checks can be performed on them. The program should be able to verify the following cases:

### Current dosage check:

When a treatment is given, the bounds of the estimated blood concentration for the current dosage will systematically be checked for dangerousness and inefficiency alerts.

### Sample check:

When one or more blood sample are also provided, their values will be checked against the surrounding percentiles to verify the likelihood of the measure, it is not so rare that an error occurs in the measuring process (wrong time or day written, samples switching between patients, etc).

The results will be provided as a one file HTML report, with the optional administrative data on top, followed by informations about the drug model chosen by the program, and then the results of the different computations. An adjustment computation is asked for all requests, so a graph representing each returned adjustment along with informations should be present in every report, but it is possible that no dosage is found, then this part would be empty. If a treatment is given, the corresponding predicted curve is shown along with percentiles and the measured samples if provided.

The report will be created in a way that allows it to be easily translated: all texts parts in the report are members or functions of a language class, that need to be derived for each supported language. The method responsible for creating the HTML will then concatenates the file's structure and the texts, inserted by calling those members and methods from the good language class, provided by a factory. Typical default sentences would look like: « Here are the (XX) available dosages found and their predicted curves » or « Warning: the provided blood sample taken at (*date*) (exceeds the 95th / is below the 5th) percentile in the population, its value might be wrong. » Which sentence are present and their final value will be determined after receiving and analyzing the computation results. The only language class created for this project is the english one, serving as default class, but the structure was conceived so one can easily add translated words and sentences to match another language tag in the input files.

The output of the core for the prediction curves and percentiles being series of points, a plotting library is used to draw them. They are then exported as images in PNG format, and finally encoded as base64 and inserted directly in an `<img/>` tag in the HTML, to keep the results in one file.

Researches have been first made on gnuplot, which can plot multiple curve types on one graph and is able to fill the interval between curves with different colors, so it seems to be able to draw the same kind of graphs than the GUI version, which are pretty nice looking. It can also produce PNG images to be inserted directly in the HTML file.

The CSS for the results's layout will be integrated within a `<style>` tag to match the one file criteria.

## IV. Implementation

*Note:* When referring later to classes of the existing Tucuxi code, they will be prefixed with their namespace, like in the code, to clearly distinguish between the existing and created code. Those namespaces are: « Core:: », « Query:: » and « Common:: »

This section will present the different classes created for this project, in the flow order of the program.

### **tucuxpert.cpp:**

This is a single .cpp file serving as program entry point, which is in charge of the command line interface handling. It will just parse the supplied arguments and check that the required ones are present.

The defined arguments for this program are the following:

- The input XML file path (required), provided with -i or --input,
- The output HTML file path (required), provided with -o or --output,
- A drug files folder (optional) provided with -d or --drugs, it defaults to a folder called « drug files », which is expected in the same folder as the executable.
- A help text, available with -h or --help.

If the required arguments are missing, the help text is displayed and the program exits.

Once the arguments parsed, it will call the « process controller » (TucuxpertHandler), to proceed with all main steps of the program.

### **TucuxpertHandler:**

This class has only one method, which takes the parsed arguments as strings, and will carry out all successive operations to obtain the results. It uses an XpertData object (details below) to store and transmit the different results between the steps. Those steps are the following:

- Creating a Core::DrugModelRepository component and setting the drug model folder, either with the provided path or the default one. This Core::DrugModelRepository is available in other parts of the code from a Common::ComponentManager singleton.
- Calling the QueryCreator, which will parse the input file, select a compatible drug model and create the requests depending on the input file's content. Its results are inserted in the XpertData Object. Since the QueryCreator may fail in many ways, TucuxpertHandler will terminate the execution in case of a bad return code.
- Calling the Query::QueryComputer class, which will execute all the the previously created queries and set the response object with the results,



- Calling the `HtmlExporter`, which will fill the HTML structure with the results and the generated graphs and save it to the disk at the specified location.
- Removing the `Core::DrugModelRepository` component.

## XpertData:

This class is meant to store the data that must be transferred between the main parts of the process. It has the following members:

- `m_lang`: a string defining the language to use, set by the `QueryCreator` from the XML.
- `m_drugModelResult`: a structure containing the drug model selected by the `DrugModelSelector`, and the results of its constraints evaluation.
- `m_queries`: a `Query::ComputingQuery` object, containing the requests to be executed by the core. This field is set by the `RequestCreator` class.
- `m_predictionType`: this field is an enum representing the type of computation performed (*population*, *a priori* or *a posteriori*) and is filled by the `RequestCreator`.
- `m_drugTreatment`: this is a pointer on a `DrugTreatment` object that contains information extracted from the `drugTreatment` tag in the XML. Since this object is dynamically allocated by the method creating it, it is stored in a `unique_ptr` to be deleted automatically with the `XpertData` object.
- `m_responses`: a `Query::ComputingQueryResponse` object that will be filled by the core with the results

This class should also contain the `AdminDatas`, the data classes to contain them have been created but at the time of this writing their parsing have still not been implemented, being a non-essential feature.

## QueryCreator:

This class is derived from the existing XML parser (`Query::QueryImport`), to benefit from its existing methods. Like its parent class, `QueryCreator` offers 2 public methods to load the XML, from a file or from a string. Currently only the method reading a file is used by `TucuxpertHandler`, the other might be useful later in case of integration with the REST server for example.

After creating a `Common::XmlDocument` object with the content of the input file or string, those 2 methods call the `createQuery` protected method, which uses the `importDocument` method from the parent class to create a `Query::QueryData` object, containing the same information in a more exploitable form. Since this `Query::QueryData` object is not needed anymore once the `Core::DrugTreatment` object have been extracted and stored in the `XpertData`, the other interesting informations must be retrieved, like the language or the query id. The first is stored in `m_lang`, and the second is set as the id of `m_queries`. Then, a `RequestCreator` object is called to fill `m_queries` with the appropriate `Core::ComputingRequest` objects. This class was first meant to return a `Query::QueryData` object to be transformed to a `Core::ComputingRequest` by another existing class, but since the `Core::DrugDomainConstraintsEvaluator` required anyway a `Core::DrugTreatment` to evaluate the drug models, the `Core::ComputingRequest` could be created directly, avoiding an object translation.

## RequestCreator:

This class has only one method, `createRequests`, which is responsible to determine what computation should be made by the core. To do this, it needs at least a drug provided in the `drugTreatment` XML field, it will terminate if none is found. TucuXpert presently supports only one drug at a time, if more than one drug is provided as input, only the first will be considered. This could be extended in the future by adding a loop to create requests for each provided drug, it would also need to change a little `XpertData` to store information for multiple drugs and the output generator to get the right objects, since some will be in multiple copies.

Once a drug have been found, the corresponding `Core::DrugTreatment` must be extracted, with the help of the `Query::QueryToCoreExtractor::extractDrugTreatment` method. This `Core::DrugTreatment` is specific to a drug (it can be confusing since all drugs are contained in a `drugTreatment` tag in the XML), and must be embedded in each request for the needs of the computing core. Since the `extractDrugTreatment` method returns a dynamically allocated object that needs to live longer than the `RequestCreator` class, its deleting is entrusted to the `XpertData` object via a `unique_ptr`. If the `Core::DrugTreatment` could not be extracted, the program will be stopped.

After the `Core::DrugTreatment` is extracted, an appropriate `Core::DrugModel` needs to be selected. This is performed by using a `DrugModelSelector`, that will analyze the compatibility of the provided covariates with the `Core::DrugModel`'s constraints. The best candidate along with the covariates evaluation results are set in `XpertData`.

Then, some boolean flags are set to determine what information were provided. This is performed by checking the size of some vectors in the `Query::QueryData` object extracted by the `QueryCreator`. These flags report if a treatment, samples, covariates and targets were supplied, they will allow to decide what request will be created, and impact some parameters. A last check is performed after setting these flags, the program will exit if samples were provided without a drug treatment, which makes no sense. After this point the request creation should not fail.

Some dates must have been used for the next part, and a small problem has been noticed while working with them: the `Common::DateTime` class default constructor creates the current date with the UTC time, so all variables that needs the current time are taking their values from a constant variable called `now`, this allows to adjust once the time zone shift or to provide a fixed time from the command line to make repeatable tests, but this has not yet been implemented.

For the request creation, some parameters have to be created first, the most important are:

- the number of points per hour asked for the computations, set to 20 for all requests.
- the prediction type, which is set to « population » when the covariates and samples flags are false (if sample is false, treatment is also false), « apriori » when no samples but some covariates are provided, and « aposteriori » otherwise (which covers all cases when samples are provided.)
- the targets option, set to use the provided or default targets depending on the targets flag, and used for all requests.
- the begin and end dates, there is 2 different version of each, one for the adjustments requests and one for the concentration prediction and the percentiles requests. For the adjustments, the begin time is used exactly by the core, so in case of an existing treatment, it is best to start the adjustment at the begin of an interval. To do this, the intakes in the next 10 days are retrieved with a low precision (1 point per hour) to cover the weekly dosages, and the date of the first intake superior to the current date is set as the adjustment start time. If no treatment or no next intake found, the default value is based on the `now` constant, and is set to the next round hour if there's more than 30 minutes left to

reach it, and to the next round hour +1 if less than 30min, to let the time to analyze the results and prepare one of the suggested treatment. Once the start date is determined, the end adjustment date is set to start + 7 days. For the concentration prediction and percentiles times, the start time will automatically be rounded to the begin of the interval containing it, so it doesn't need to be adjusted. the default value is now - 1day, but is set to the oldest sample's date found in the 2 days preceding now if any, to include them in the predicted curves. The end value being not rounded, it is set to an interval end that is at least 24h after the adjustment start, or just 24h after it if no interval could be extracted.

Each desired computation is embedded in a `Core::ComputingRequest` object, that always contains an id, a `Core::DrugModel`, a `Core::DrugTreatment`, and a subclass of `Core::ComputingTrait`, which represents this computation and contains its required parameters. The requests are created by TucuXpert with the following computing traits:

- A `Core::ComputingTraitAdjustment` in all cases, asking for the best dosage for each interval and returning the predicted curve points and the corresponding dosage history.
- A `Core::ComputingTraitConcentration` and a `Core::ComputingTraitPercentiles` when the treatment flag is true, those requests are always performed together, and there should be no reason that one fails without the other. This property is sometimes used in the code to avoid extensive checks. The first object computes the predicted blood concentration over the asked period, and the second computes the associated percentiles, at the ranks 5, 25, 75 and 95.
- A `Core::ComputingTraitAtMeasures` when the sample flag is true, which will just return the predicted blood concentration at the precise sample time, since the `Core::ComputingTraitConcentration` gives points every 5 minutes. This value could be interpolated, but since this object exists and returns a precise value it has been used.

## DrugModelSelector:

This class selects an appropriate `Core::DrugModel` to use with the requests. It must first get the `Core::DrugModelRepository` from the `Common::ComponentManager` singleton, and then get from it an array of `Core::DrugModel*` with all parsed drug files having the desired drugId. If no model is found, the program terminates.

Then, the constraints of each model are evaluated against the provided covariates, with the help of a `Core::DrugDomainConstraintsEvaluator`, and the results of each evaluation are stored. These results can be of 4 types: compatible, partially compatible, incompatible, or computation error. The models having any result of the last 2 types are discarded, and the « best » is chosen with the following criteria: the model with the highest number of fully compatible constraints is selected, and in case of equality the number of partially compatible constraints is compared. If also equals, the first model found will be chosen. A first approach based on the ratio of the number of compatible over partially compatible constraints was conceived, but for an equal number of fully compatible constraints, it favored the model with the least partially compatible constraints, while the desired goal was to have the most accurate model, hence the most constraints. The final choice is still arbitrary and would need to be checked with a pharmacologist.

When the model is chosen, its address is copied in the `DrugModelResult` structure passed as parameter, along with the corresponding evaluation results.

## HtmlCreator:

The only public method of this class receives the XpertData object and the output path to write the HTML file as parameters, and is responsible to extract and display all useful information, mainly from the Query::ComputingQueryResponse object returned by the core, but also on the drug model selected and the administrative data when their parsing will be implemented. It uses the ILanguage class that automatically provides texts in the language corresponding to the language tag, if the corresponding subclass exists, it defaults to english otherwise.

This class also uses a GraphCreator to generate the graphs that will be inserted.

The HTML structure is written as strings, mainly using <section> tags to separate the different parts, <table> tags to present informations and <img> tags to embed the graphs. The structure is concatenated with results and text parts, in the following order:

- The HTML headers with the CSS rules included in a <style> tag and the page title,
- The query ID,
- Informations on the drug model used: its name, the drug concerned, the constraints and their results, and the covariates defined with their default, provided and used (if available) values.
- The predicted blood concentration and percentiles graph if any, along with the results of checks performed. This part is absent if no treatment is provided in input.
- The different adjustments suggested, including for each of them a graph showing the adjustment and the current dosage if any, the targets evaluation results and the dosage(s) indicated for this adjustment.
- The end of the HTML structure, including a small javascript function to enlarge the images when clicked. This is made simply by toggling the CSS property max-width from 100% to none.

The results check is performed directly in this class, being simple verification loops. The toxicity and inefficacy limits were planned to be checked only if the target's units are compatible with the result's one (*i.e.* a concentration), some drug models having targets in other units like hours, but calling the UnitManager::isCompatible method creates a linker error, even if other methods in UnitManager are working properly. This being one of the last implemented feature with no time left to check this error, it has been commented out. The samples check is performed for every sample provided within the limits of the prediction asked if any. Since the check for the upper and lower bounds are very similar, a private function has been created with a boolean parameter to indicate the concerned bound. This function returns an empty string if everything is ok, and a full warning tag if a sample is out of bounds, so it can be concatenated directly to the HTML.

The image insertion is made by encoding the PNG images as base64 and inserting this string directly in the src property of the image tag. The exact syntax is: src="data:image/png;base64, <image string>". This was possible entirely by using the QByteArray class, which can read a file, convert it to base64 and output it as a string. This adds a dependency on Qt but since the whole project was initially provided with either Qt or virtualStudio project files, this only prevents to launch the project with virtualStudio, but can easily be changed by finding another good free base64 library.

## GraphCreator:

This class is responsible for generating the graphs from the adjustments, concentration prediction and percentile's computed data, and to plot the measured samples within the prediction. To do this, multiple libraries options have been explored, a lot of libraries found are outdated or just wrappers calling other languages (mainly python and javascript), researches have been focused on finding a full c++ library that can draw multiple curves on the same graph, has filling options to represent the percentiles, allows to change the axes labels, and fits in the minimal number of files. Here are some of the considered libraries:

- gnuplot: comes as a compiled program with all options, to call via an interface that pipes commands, but the interface needs c++17 and my compiler doesn't support it. This was the first choice but has been abandoned.
- cplot: the curves are plotted from c++, but in json format and it needs javascript tools to read it (react) the output was very nice tough, the curves are zoomable
- antigrain geometry: found good reviews, but is old, even their website is broken.
- wxmathplot : seems old, and didn't found examples with curves filling
- dislin: output is not very nice, also didn't found examples to fill between curves
- matplotlib-cpp: a wrapper around python matplotlib, produces a very nice output, and quite every machine has python installed, this would have been the third choice
- mathGL: seems good with all options, but depends on others libs like boost. This would have been the second choice.
- QCustomPlot: a c++ wrapper around Qt's graphical classes that fits in one .h and one .cpp file, produces a very nice output easily, and since the project already uses Qt it just required to include some Qt modules in the .pro file, and creating an empty QApplication at the program start so the graphical QWidget could be created. This option has been selected, its only bad point is that the license is GPL, so it can be used in the scope of this bachelor work, but another licence type has to be enquired or this library should be changed if one day this program is to sell.

The main work performed in this class is to convert between dates and the relative time given in results, the drawing part being fairly easy with the library: every Core::CycleData object in the results contains the points for its cycle expressing the time points (x axis) as the relative time in hour since the beginning of the cycle; so for each prediction, adjustment or percentile, the arrays of each cycle have to be concatenated, and for the x axis the last value of an array has to be added to all elements in the next one. Additionally, when an adjustment and a concentration prediction are present, the time shift between their beginning has to be calculated to plot them on the same graph in a coherent way. Two methods have been created to concatenate these arrays, one for the x and y axes that can also take a starting time shift, and one only for the y axis, because for percentiles the x axis computed for the concentration prediction is reused.

Another point requiring some computations is the creation of the x axis legend, to indicate time in days while data is given as relative time, so the default numbering is the number of hour since the graph begin, which is not very readable. It requires to find the graph's total duration, the offset between the graph begin and the first following midnight, and to set « ticks » with the string to display and the relative time point it belongs to for the total duration of the graph. The first version only displayed full days, but it was spaced and it was easy to add a tick every 6h for a better readability.

About the graph creation, the same `QCustomPlot` object (the base object of the library) is used for all graphs, since it allows to add and remove curves from it, this property is used to plot only once the adjustment curve if present, in a top layer so it will not be covered by the percentiles added after. Then each adjustment is added with its time shift to a new graph, the axes are adjusted automatically, the whole graph is exported as a PNG file, and the adjustment graph is removed, allowing for another one or percentiles to be drawn. For the percentiles, each of them is added to the graph, and 2 of them are set to have a filling color that goes up to the 2 others.

If sample have been provided, their value is converted in the result's unit and their date is converted to a relative time since the beginning of the graph. They are then plotted with an empty line style, so the points are not joined. Samples older than the graph begin are not plotted.

Different colors are used for the concentration prediction and the percentiles, depending on the prediction type. These colors fits the current Tucuxi color code.

The height of all images is 600px, but their width is adjusted depending on the number of days represented, with this small formula:  $(nbOfDays + 5) / 3 * 400$  px. This corresponds to 800px minimum for 1, 2, and 3 days, then adds 400px and so on every 3 days, to keep a readable image.

The class also assumes that the `CycleData`'s points, which are in a vector of vector, are all in the vector contained in the first cell of the outer vector.

## ILanguage:

This class is meant to be subclassed for all supported output languages. It defines all used pieces of text as members and phrases as methods, to allow parameters that might be inserted at different places in the sentence depending on the language. It has a protected constructor, so only subclasses can initialize it with the values in their language, and they must also provide the overriden methods. It also has a static `getLang` method that takes the language string as input and returns a pointer on the corresponding class dynamically allocated, so the caller is responsible for its deletion. This is due to the fact that the dynamic linking with a subclass works only via pointers and references in c++.

## EnLanguage:

This is just the default language class implementing `ILanguage` with english, returned as the default class by the `ILanguage::getLang` method if the language string is not known.

## Tools:

This class contains only static methods, mainly to print `Common::DateTime` objects, and a templated method to extract a specific response object with its id from the responses array.



## V. Examples of outputs

(pdf can be zoomed and images are available as appendices.)

### Simple case with no treatment:

#### TucuXpert Results

11560670\_ch.tucuxi.vancomycin.colin2019\_0

##### drug model:

drug used: vancomycin

drug model used: ch.tucuxi.vancomycin.colin2019.accurate

##### constraints for this model:

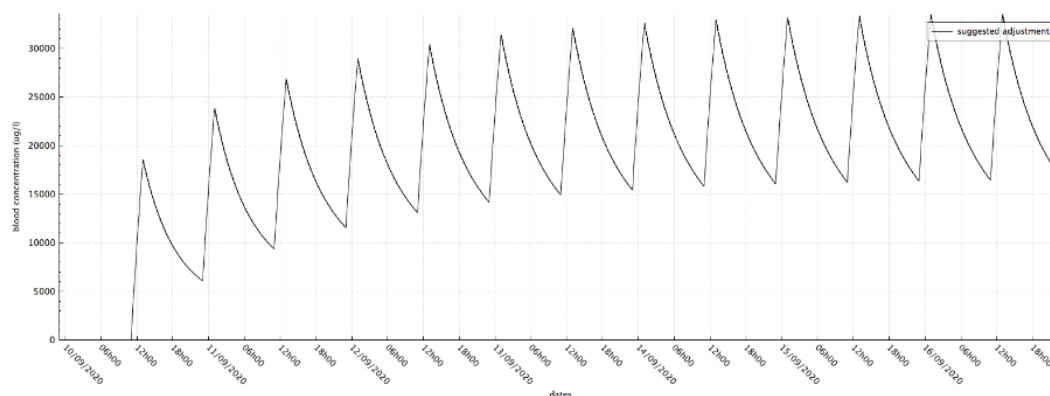
The age shall be greater or equal to 18	compatible
The age shall be greater or smaller or equal to 77	compatible
The weight should be in the [50,130] margin	compatible

##### covariates defined:

name:	default value:	provided value:
bodyweight	70.000000 kg	56.9 kg
sex	0.500000 -	1 -
age	35.000000 y	1973-06-20 00:00:00 -
scr	73.400000 umol/l	multiple values
CLcr	70.000000 ml/min	computed
creatinine	0.830000 mg/l	computed
ga	34.000000 w	missing
pna	12775.000000 d	1973-06-20 00:00:00 -
haem	0.000000	missing

The type of prediction used is: a priori

##### suggested adjustment:



##### data for this adjustment:

###### target evaluation:

global score: 0.916248

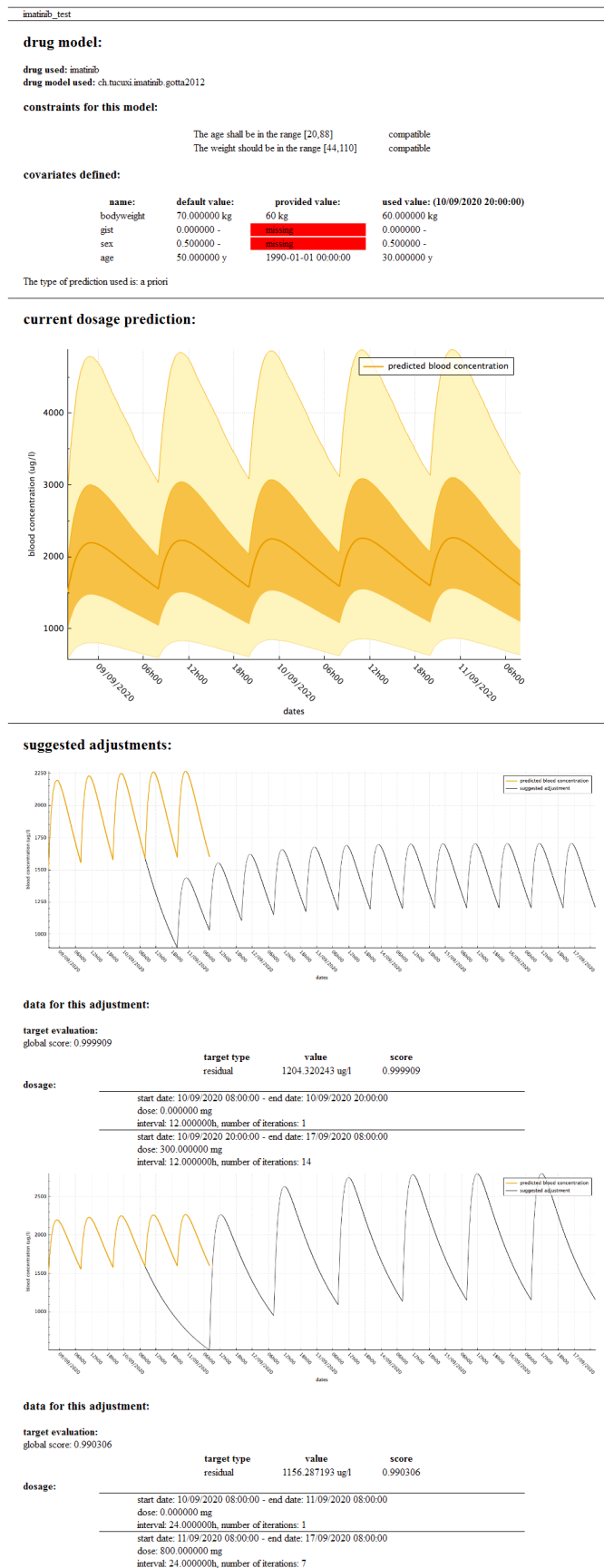
target type	value	score
auc24DividedByMic	565.217993 h	0.916248

##### dosage:

start date: 09/09/2020 23:00:00 - end date: 10/09/2020 11:00:00
dose: 0.000000 mg
interval: 12.000000h, number of iterations: 1
start date: 10/09/2020 11:00:00 - end date: 16/09/2020 23:00:00
dose: 750.000000 mg, infusion time: 120.000000 min
interval: 12.000000h, number of iterations: 14

## Case with a treatment and no samples, showing multiple adjustments:

### TucuXpert Results





## Case with samples including one outside the range:

### TucuXpert Results

11560670\_ch.tucuxi.vancomycin.colin2019\_0

#### drug model:

drug used: vancomycin

drug model used: ch.tucuxi.vancomycin.colin2019.accurate

#### constraints for this model:

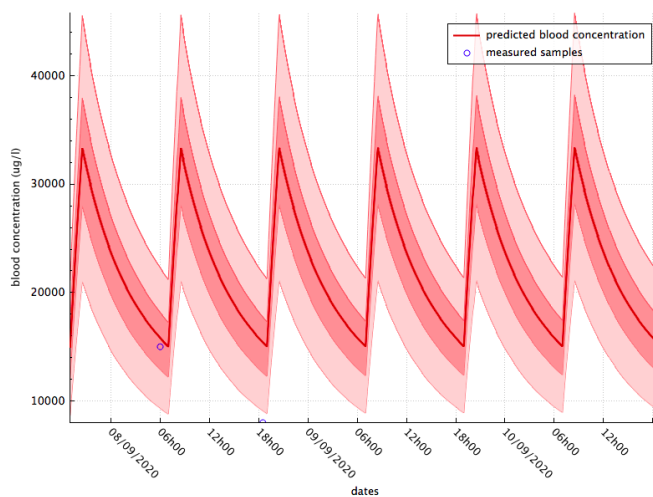
The age shall be greater or equal to 18 compatible  
 The age shall be greater or smaller or equal to 77 compatible  
 The weight should be in the [50,130] margin compatible

#### covariates defined:

name:	default value:	provided value:	used value: (10/09/2020 12:00:00)
bodyweight	70.000000 kg	56.9 kg	56.900000 kg
sex	0.500000 -	1 -	1.000000 -
age	35.000000 y	1973-06-20 00:00:00 -	47.000000 y
scr	73.400000 umol/l	multiple values	92.000000 umol/l
CLcr	70.000000 ml/min	computed	6254.099069 ml/min
creatinine	0.830000 mg/l	computed	1.040724 mg/l
ga	34.000000 w	missing	34.000000 w
pna	12775.000000 d	1973-06-20 00:00:00 -	17249.000000 d
haem	0.000000	missing	0.000000

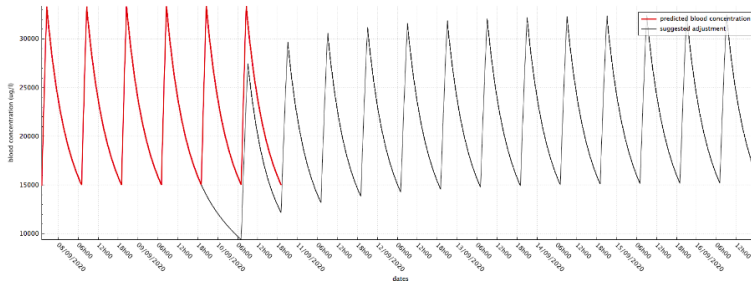
The type of prediction used is: a posteriori

#### current dosage prediction:



Warning: the sample taken at: 08/09/2020 18:30:00 is below the 5th percentile, its value might be wrong.

#### suggested adjustment:



#### data for this adjustment:

target evaluation:  
global score: 0.978744

target type	value	score
auc24DividedByMic	531.856242 h	0.978744

#### dosage:

start date: 09/09/2020 19:00:00 - end date: 10/09/2020 07:00:00  
 dose: 0.000000 mg  
 interval: 12.000000h, number of iterations: 1  
 start date: 10/09/2020 07:00:00 - end date: 16/09/2020 19:00:00  
 dose: 750.000000 mg, infusion time: 120.000000 min  
 interval: 12.000000h, number of iterations: 14

## VI. Difficulties encountered

Since designing a solution to the problem required to deal with a lot of existing classes, it was very hard in the beginning to get a glimpse of the existing project and to find what to use and how, it felt similar to doing a big puzzle with only having someone describing you the final result, and being able to look at only one piece at a time.

The graphs creation took longer than expected due to problems with gnuplot, and other solution had to be searched for. The different computations to convert and adjust the dates for the graph plotting were also pretty time-taking.

Getting the data for the output was painful, data being deeply nested within the objects results, this required to dig in the code very often, and the structure is not regular: once there's a getter, the other time members are public. Cumulating the fact that Qt creator doesn't recognizes unique\_ptr's operators \* and ->, which causes the autocomplete to fail. Sometimes there's also inherited methods with no sense (for example adjustmentData->getData returns nothing, need to use adjustmentData-> getAdjustments()[i].m\_data )

I had not written c++ for more than 1 year before this project, so I had some surprises, notably when creating the ILanguage interface, for which I had to change a little my plans (but the result is the same).

There was also some problems with existing parts, which created weird results that took some time to analyze, most of them are now fixed. These includes :

- the constraints extractor used to put default values for hard constraints when a hard constraint should fail if not provided
- the drugModelConstraintsEvaluator returned a computation error for a lot of models,
- there was also errors in intervals extraction depending on the query time, outputting « zero interval » and sometimes failing.
- adding samples to the input file makes the percentiles computation produce a lot of errors (division by zero, negative concentrations, Applying Eta to Parameter CL makes it not a number), but still works, it just takes much more time to complete. This error is still not fixed at the time of this writing.
- a getter returning empty objects while the debugger shows that it exists, and accessing its public member works.

## VII. Possible improvements

- More data could be printed in the output HTML, like the specific targets, the formulation and route for the suggested adjustments or the statistics for the last predicted cycle.
- The admin datas still needs to be parsed and included in the HTML
- The concentration prediction bounds check needs to be debuged, and a way to evaluate targets with incompatible units could be found, since the core can use this information.
- `Common::DateTime` uses the UTC time, need to find a way to get the local time.
- A date parameter could be provided in input to set a fixed current time for tests.
- Doubles are printed with 6digits precision in the HTML, an ostreamstream could be used with the `precision()` method,
- The HTML could be properly indented, currently it just has line feeds,
- The suggested adjustments always return an interval with a first dose of 0, this needs to be checked.
- For now we suppose that there is only one drug in the `drugTreatment`, others are ignored, but the structure could already be changed to handle multiple drugs.
- Create some units tests to certify the good behavior of the program.

# Bibliography

Tucuxi part:

- Nadir Benallal, Bachelor thesis, 2018
- The Tucuxi documentation
- Yann Thoma, the alive Tucuxi encyclopedia

C++ part:

- [cppreference.com](http://cppreference.com)
- [stackoverflow.com](http://stackoverflow.com)
- [www.qcustomplot.com/index.php/demos](http://www.qcustomplot.com/index.php/demos)